# Module 7: Source Code Management

**Git, Branching, and GitOps for Infrastructure as Code**

---

# Learning Objectives

By the end of this module you will understand:

- Why version control is essential for Infrastructure as Code
- What GitOps is and how it applies to Terraform
- How `.gitignore` protects sensitive Terraform files
- The branching workflow for team collaboration
- How Pull Requests enable code review

---

# Why Version Control for Infrastructure?

**The Problem Without Version Control:**

Imagine managing Terraform without Git:

- `main.tf` (which version is this?)
- `main_v2.tf` (what changed?)
- `main_final.tf` (really final?)
- `main_final_ACTUALLY_FINAL.tf` (not really!)

**Questions you can't answer:**

- Who changed the security group rule last week?
- What did the infrastructure look like 2 months ago?
- How do you undo a mistake?
- How do multiple people work without overwriting each other?

---

# What Version Control Provides

**Git gives you:**

- **Version history** - every change tracked with who, what, when, why
- **Collaboration** - multiple people work in parallel safely
- **Code review** - changes reviewed before applied
- **Audit trail** - know exactly what changed and when
- **Rollback capability** - undo mistakes easily
- **Branching** - experiment safely without affecting production

---

# What is GitOps?

**GitOps is a way of managing infrastructure where:**

> Git is the single source of truth for your infrastructure

**The core idea:**

- Your infrastructure is defined in code (Terraform files)
- That code lives in a Git repository
- Any change to infrastructure starts with a change in Git
- Git history becomes your infrastructure history

---

# GitOps Workflow

**How changes flow in GitOps:**

```
1. Developer writes Terraform code
         |
         v
2. Code is committed to Git repository
         |
         v
3. Pull Request created for review
         |
         v
4. Team reviews and approves changes
         |
         v
5. Code merged to main branch
         |
         v
6. Terraform applies the changes to infrastructure
```

This workflow ensures every infrastructure change is:

- Reviewed before applied
- Tracked in version history
- Reversible if needed

---

# Benefits of GitOps for Terraform

**Why use Git with Terraform?**

| Benefit | What It Means |
| --- | --- |

| Benefit | What It Means |
|---|---|
| Auditability | Complete history of who changed what and when |
| Reproducibility | Anyone can recreate the infrastructure from code |
| Collaboration | Teams can work together without conflicts |
| Review | Changes are checked before going to production |
| Rollback | Easy to revert to a previous working version |

---

# How Git and Terraform Work Together

**The relationship:**

- **Terraform files** (.tf) define your infrastructure
- **Git** tracks changes to those files over time
- **Remote repository** (GitHub) stores the code for the team
- **Branches** let you work on changes without affecting others
- **Pull Requests** let the team review before applying

**Key insight:** Treat your Terraform code with the same care as application code!

---

# The .gitignore File

**What is .gitignore?**

A special file that tells Git which files to ignore (not track).

**Why it matters for Terraform:**

Terraform creates several files that should NOT be committed:

- State files (contain sensitive infrastructure data)
- Downloaded providers (large, regenerated automatically)
- Variable files with actual values (may contain secrets)

The `.gitignore` file protects you from accidentally committing these files.

---

# What .gitignore Excludes for Terraform

**Files that should be ignored:**

| File/Directory | Why Ignore It |
|---|---|
| `.terraform/` | Downloaded providers, regenerated on init |

| File/Directory | Why Ignore It |
| --- | --- |
| `*.tfstate` | Contains sensitive data, should be stored remotely |
| `*.tfstate.*` | Backup state files |
| `*.tfvars` | May contain passwords and secrets |
| `override.tf` | Local overrides not meant for the team |

**You will create this file in Lab 6.**

---

# What to Commit to Git

**DO commit these files:**

- `.tf` files - All Terraform configuration
- `.gitignore` file - Tells Git what to ignore
- `terraform.tfvars.example` - Template showing what variables are needed
- `.terraform.lock.hcl` - Locks provider versions for consistency
- `README.md` - Documentation for your project

---

# What NOT to Commit

**DO NOT commit these files:**

- `terraform.tfstate` - Contains sensitive infrastructure data
- `.terraform/` directory - Downloaded providers, temporary files
- `*.tfvars` with real values - May contain passwords and API keys
- Any file with secrets - Passwords, credentials, private keys

**Remember:** If it contains secrets or is auto-generated, don't commit it!

---

# Branching Strategy Overview

**Why use branches?**

Branches let you:

- Work on changes without affecting the main code
- Have multiple people work at the same time
- Review changes before they go live
- Keep a stable main branch

---

# The Three-Branch Workflow

**A simple branching strategy:**

```
main (production-ready code)
  ^
  |
develop (integration branch)
  ^
  |
feature/my-change (your work)
```

**How it works:**

1. `main` - Always contains stable, working code
2. `develop` - Where features are integrated and tested
3. `feature/*` - Where you do your actual work

---

# Branch Flow

**The typical workflow:**

1. Start from `develop` branch
2. Create a new `feature` branch for your work
3. Make changes and commit them
4. Push your branch to the remote repository
5. Create a Pull Request to merge into `develop`
6. Team reviews your changes
7. After approval, merge into `develop`
8. Delete the feature branch (it's done!)

---

# What is a Pull Request?

**A Pull Request (PR) is:**

- A request to merge your branch into another branch
- An opportunity for the team to review your code
- A discussion forum about the change
- A record of what changed and why

**Also called:** Merge Request (MR) in some systems like GitLab

---

# Why Pull Requests Matter

**Pull Requests provide:**

| Benefit | Description |
|---|---|
| Code Review | Others check your work before it goes live |
| Discussion | Team can ask questions and suggest improvements |
| Quality Control | Catch mistakes before they affect infrastructure |
| Documentation | The PR description explains what changed and why |
| Approval Workflow | Require sign-off before changes are applied |

---

# The Pull Request Process

**How a Pull Request works:**

1. You push your feature branch to the remote repository
2. You create a Pull Request (usually through the web interface)
3. You describe what changes you made and why
4. Team members review the changes
5. They may request modifications
6. Once approved, you merge the Pull Request
7. The feature branch can be deleted

**You will practice this in Lab 6.**

---

# Code Review Basics

**What to look for when reviewing Terraform code:**

**Security:**

- No hardcoded secrets or credentials
- Encryption enabled where appropriate
- Access properly restricted

**Best Practices:**

- Resources have meaningful names
- Variables used instead of hardcoded values
- Proper tags applied

**Quality:**

- Code is formatted consistently
- Changes make sense for the stated purpose

---

# Key Takeaways

1. **Version control is essential for IaC** - Track every infrastructure change
2. **GitOps uses Git as the source of truth** - All changes flow through Git
3. **Use .gitignore to protect secrets** - Never commit state files or credentials
4. **Branches enable safe collaboration** - Work without affecting others
5. **Pull Requests enable review** - Catch issues before production
6. **Treat infrastructure as code** - Same practices as software development

**Next:** Practice this workflow hands-on in Lab 6!

---

# Knowledge Check

**Question 1:** Which files should you NOT commit to Git in a Terraform project?

A) main.tf and variables.tf B) terraform.tfstate and .terraform/ C) .gitignore and README.md D) outputs.tf and providers.tf

---

# Knowledge Check

**Question 2:** What is the main purpose of a Pull Request?

A) To automatically deploy infrastructure B) To allow code review before merging changes C) To delete old branches D) To run Terraform commands